SPECIAL ISSUE PAPER

Deformable polygonal agents in crowd simulation

Thomas Pitiot*, David Cazier, Thomas Jund, Arash Habibi and Pierre Kraemer
ICube, Université de Strasbourg, CNRS UMR 7357, Geometry and Computer Graphics Group, Illkirch-Graffenstaden, France

ABSTRACT

To produce impressive virtual worlds, real-time crowd simulations require large and detailed scenes populated by agents with complex shapes and geometry. For efficiency reasons, these agents are usually approximated by point-like representations to optimize the performances of collision avoidance and interactions between agents. This paper addresses the issue of handling deformable polygonal agents with arbitrary shapes in real time crowd simulations. The proposed multiresolution framework supports environments with arbitrary topologies and provides tools for efficient proximity queries.

KEYWORDS
crowd simulation; collision detection; virtual worlds; deformable entities; particle tracking

*Correspondence
Thomas Pitiot, ICube, Université de Strasbourg, CNRS UMR 7357, Geometry and Computer Graphics Group, Illkirch-Graffenstaden, France.
E-mail: pitiot@unistra.fr

1. INTRODUCTION

Crowd simulation is an important tool for the production of virtual worlds for entertainment or architectural and urban planning. While walking along their own paths, virtual agents have to adapt their trajectories to avoid obstacles and other agents. To achieve real time performances, agents and interacting objects are commonly represented by points. Recent works address the interactions of agents with large objects whose shapes are approximated by disk or simple convex polygons. Handling interactions with more complex entities – such as articulated or deformable mobiles – is still an open issue.

In this paper, we address the real-time simulation of crowds that include agents with complex and deformable shapes. Such agents are modeled as surface meshes with arbitrary shapes and topologies (figure 1). Their deformations and animations are controlled through polyhedral cages bounding them. The footprints of these cages on the ground surface form possibly concave polygons that are used for collision avoidance and proximity queries. In the following, the term polygonal agents refers to the footprints of these cages.

The main issue addressed in this work is the management of the interactions between point-like and polygonal agents moving in dynamic environments. These latter agents can be complex non-planar urban environments filled with fixed obstacles (such as buildings, bridges, roads or trees). This work is an extension of [1] that introduced a unified structure for crowd simulation. This multiresolution data structure is used for the representation and rendering of the environment, for the tracking of point-like agents and to optimize proximity queries.

Figure 2 schematizes the setting of the scenarios we aim to manage. The environment is modeled as a cellular decomposition whose cells (faces or edges) may be marked as obstacles. The agents move in this environment and avoid collision with the obstacles and other agents.

The first contribution is a multiresolution and adaptive registration of agents in the environment. The multiresolution approach allows the limitation of the number of agents considered in interaction. It allows agents to query their surrounding obstacles and neighbors in constant time even in dense scenarios. The long range registration of agents with high velocities is also supported.

The second contribution is the definition of interaction forces between polygonal shapes. We propose an adaptive evaluation of these forces through an edge-based decomposition of force fields. The agents sum the forces generated by other agents in their immediate vicinity. Our framework naturally selects the edges of the polygonal agents that actually contribute thus avoiding unnecessary computations. The resulting forces require agents to move away from each other and from fixed obstacles.

The interaction forces are computed independently for each vertex of polygonal agents. To maintain their shapes and allow physically plausible deformations, internal
Deformable polygonal agents in crowd simulation

Figure 1. An agent is modeled by a 3D mesh. The control cage supports the physical model and control the deformations. Its footprint is used for collision avoidance and proximity queries.

cohesion forces must be taken into account. We use the widespread shape matching model [2] for that purpose. Other physical models can be used as long as the gathering of forces remains consistent with the registration system.

The paper is organized as follows: section 2 reviews related work. Section 3 describes the force that constrain the moves and deformations of agents. Section 4 details the registration process. Section 5 presents experimental results and an analysis of time complexity.

2. RELATED WORK

Many studies address the simulation of crowds composed of autonomous agents and larger objects like vehicles or dynamic obstacles. But few of them handle the interactions between agents and dynamic objects. In [3,4] agents and vehicles are processed in separate ways. In [5,6] authors focus on motion planning in dynamic environments and improve the usual navigation meshes by allowing jump on moving obstacles.

Interactions between autonomous agents and rigid objects is addressed in [7]. The velocities of agents are constrained by the moves of other agents through reciprocal velocity obstacles [8]. Forces produced by the Newtonian dynamics of objects are used to add constraints to the velocities. The resulting trajectories generate physically plausible behaviors and interactions.

Real time interactions with rigid objects are supported, but the footprints of these objects are approximated by discs for collision detection with agents. Our approach is complementary. If the physical model we present is simpler, our framework handles deformable objects with large and possibly non convex footprints.

To our knowledge, interactive collision avoidance and proximity querying for deformable, possibly non convex, polygonal agents is still an open issue.

2.1. Behavior

Much of the work on the behavior of autonomous agents aims at the reproduction of phenomena such as flocking, the formation of queues or human interaction [9–11].

Rule based methods have been widely used for virtual humans. In [12] and [13] specific navigation rules apply on an agent depending on the local configuration of neighboring agents and obstacles. In [14] a regular grid is used to store information about probable paths or possible behaviors when an agent reaches some destination. These approaches offer intuitive control and the obtained behaviors are often realistic. However, the result of their combination is difficult to predict and are not compatible with the physical model used to handle the deformations of polygonal agents. Mixed approaches are possible, but beyond the scope of this paper.

Primarily used in the robotics field, the velocity-based method introduced in [15] is robust and widespread. This method has been extended to the case of multiple agents using the notion of Reciprocal Velocity Obstacle [8] and optimized in [16] to increase robustness. Recent work propose hybrid methods for long range collision avoidance. These approaches assume that the agents’ behaviors are reciprocal. That is not the case here because of the presence of large and deformable agents.

In [17], a continuous representation of the crowd density is combined with a discrete, velocity-based method to obtain more natural trajectories for groups of agents. These approaches provide necessary and sufficient conditions for a collision-free simulation in the context of fixed obstacles, but in dense crowds containing large and deformable objects, over-constrained situations may arise.

Approaches based on potential fields that define repulsive and attractive forces applied on agents, goals and obstacles [18–20] can find solutions to these situations. They have proved to be well suited for large simulations and to be able to produce lanes and flocking. Usually,
these methods use a discretization of the force fields on a grid from which the trajectories of agents are computed. These approaches are poorly compatible with large and deformable mobiles whose repulsion fields have to be updated every time steps. Our multiresolution approach offers an efficient alternative where the computed force fields are naturally subdivided according to the local density of agents.

### 2.2. Proximity Queries

The most widespread ways to implement and optimize proximity queries use data structures such as space partitioning trees [8,16,20], regular grids [13] or spatial hashing [21].

Space partitioning trees, such as kd-trees or BSP trees, suffer from high update costs and are thus not adapted to dynamic environments where the distribution of entities changes frequently. Grid-based methods and spatial hashing require the agents to register in cells that form a partition of the ground. Their efficiency is very sensitive to the chosen grid resolution.

Indeed, the registration of every agent has to be updated as it passes from one cell to another. Thus, the update cost is lowered when these changes are less frequent, i.e. when the grid is coarser.

On the other hand, the number of agents that are registered in a same cell – and thus potentially interact – increases with the cell size. Thus, the efficiency of proximity queries increases as the cell size decreases, i.e. when the grid is finer.

A multiresolution approach that adapts the size of the cells to the density of agents is a good trade-off between these two choices. The framework presented in [1] offers good performances and is flexible enough to support polygonal agents. As our works rely on this structure, we describe it further in section 4.

### 3. Handling Agents Behavior

We present a model of behavior based on two types of forces: external forces that model the interactions between agents and internal forces that control the deformations of polygonal agents.

The external forces are defined through force fields that cause the agents to move away from each other. The internal forces, based on the Shape Matching method, ensure a certain elasticity of agents, while maintaining them as close as possible from their initial shapes.

Finally, we assume that every agent follows a path that is dynamically computed by an external component. Thus, every agent has a current position and velocity, plus a preferred velocity toward its destination.

In the following, the term edge refers to an edge of a polygonal agent. The term vertex refers either to a point-like agent or to a vertex of a polygonal agent. The positions and velocities are transformed as follow: (a) for every vertex and edge, a potential field is computed and the generated forces applied to the agents in its vicinity; (b) for every polygonal agent, its Shape Matching forces are computed and these internal forces applied to its vertices; (c) for every vertex, those forces are numerically integrated to compute its new position and velocity.

#### 3.1. Interaction Forces

Every vertex generates a spherical force field. This force field decreases as a second order polynomial function of the distance. The force generated by a vertex at distance $d$ is:

$$ F = k(R - d)^2 - zd $$  \hspace{1cm} (1)

where $d$ is the first time derivative of the distance, $k$ is the stiffness of the repulsion, $z$ is its damping factor and $R$ the radius of influence. No force is applied beyond a distance of $R$.

Every polygonal agent and every obstacle generates a force field whose intensity is maximal inside the polygon and decreases with the distance to that polygon. Special attention is given to possible interpenetrations as numerical integration on discrete time steps could produce unwanted situations where some vertices lay inside these polygons.

Let $P = \{p_0, p_1, \ldots, p_{n-1}\}$ be a polygonal agent or an obstacle and let us denote $p_n = p_0$. We build a triangulation that partitions $\mathcal{P}$ such that each edge $e_i = (p_i, p_{i+1})$ is associated to the triangle $\tau_i = (p_i, p_{i+1}, p_k)$, $p_k$ being an other vertex of $\mathcal{P}$. This triangulation helps to quickly determine whether a vertex is inside a polygon or not. It is either precomputed or regularly updated depending on the intensity of the deformations.

Every edge $e_i$ generates a force field defined as follows. Let $a$ be a vertex. If $a$ lies outside $\tau_i$ then it undergoes an ellipsoidal force field – for which $p_i$ and $p_{i+1}$ form the foci – defined by:

$$ F_i = k(A - d_i - d_{i+1})^2 - zd_i - zd_{i+1} $$  \hspace{1cm} (2)

where $A$ represents the ellipsoid’s semi-major axis and $d_i$, $d_{i+1}$ represent the distance of $a$ to $p_i$ and $p_{i+1}$. The force is null when $d_i + d_{i+1}$ is greater than $A$ and the intensity is maximal on the $[p_i, p_{i+1}]$ line segment.

If $a$ is inside $\tau_i$, an interpenetration occurs. Let $a_b$ be its projection on the nearest edge of $\tau_i$. The repulsive force applied on $a$ is the value of the force field at $a_b$. As up to three edges $e_j$ may be associated to $\tau_i$, the choice of the nearest one for the projection assures the consistence of the responses along the simulation.

The direction of this force is set to the normal of the ellipse of foci $p_i$ and $p_{i+1}$ passing through $a$. This normal $\vec{n}$ is computed with:

$$ \vec{n} = p_{i+1}a + \frac{d_{i+1}}{d_i + d_{i+1}}p_ip_{i+1} $$  \hspace{1cm} (3)

The choice of ellipsoidal force fields rather than cylindrical ones facilitates the blending of the forces applied by
several edges. This blending is obtained with a simple sum of the forces. Thanks to the influence of distances in these formulations, the intensities and the directions of the normals are soundly and smoothly blended around corners. This choice also limits the computational cost of the evaluation of the force fields, as only sums of distances are required that are numerically more robust than point-to-segment distances.

Figure 3 shows the potential fields around point-like and polygonal agents. Provided with such a model, vertices avoid collisions whether they belong to point-like or polygonal agents.

### 3.2. Shape Matching

In order to maintain the initial shape of a polygonal agent through the simulation we use the Shape Matching algorithm described in [2].

Firstly, the Shape Matching algorithm stores the initial position of the deformation cage’s vertices of the polygonal agent. Then, when those vertices move during the animation process, a translation and rotation of the initial shape is computed to fit the new shape as close as possible. This computation gives a goal to each vertex that would maintain the initial shape.

Finally, we can control the plasticity of the agent through two parameters $\alpha$ and $\beta$. $\alpha$ controls the importance of following the goal given to each vertex while $\beta$ controls the freedom given to the fitting algorithm.

### 3.3. Integration

When all force fields are estimated, a classical Euler integration method is used. The position $P(t)$ and velocity $V(t)$ of a vertex of mass $m$ at time $t$ is computed with the following equation:

$$V(t + dt) = V(t) + F(t) \frac{dt}{m}$$

The norm of $V(t + dt)$ is bounded to a maximal velocity $V_{\text{max}}$, which guarantees the stability of the solution. The position is then computed as follows:

$$P(t + dt) = P(t) + V(t + dt)dt$$

### 4. PROXIMITY QUERIES

During the simulation, every agent has to sense the environment for nearby neighbors. The way these proximity queries are resolved is a crucial for the global efficiency and scalability of the simulation. We extend an existing framework that we further adapt to handle polygonal agents.

#### 4.1. Background

A unique structure is used for the representation of environments, proximity queries and path planning. The environment is modeled as a multiresolution surface mesh that allows the simulation to take place on any manifold (figure 4). The agents can move on this mesh, except across faces that were explicitly marked as obstacles (such as walls, roofs or doors).

Agents’ vertices are registered within the face they belong to. This information is updated each time a vertex crosses an edge. This information leads to efficient topological queries to retrieve the list of agents in the vicinity of a given face.

Several optimizations of this process are proposed: interacting objects are queried only within the one-ring of the agent.
current face of a vertex (implying some constraints in the faces’ shape); an adaptive subdivision of the mesh based on the density of agents allows to keep the number of agents registered in a face below a given threshold. The advantage is that even in dense scenarios, each agent only needs to consider a limited number of other agents for interactions.

4.2. Polygonal Agents

We describe here how the above mentioned framework is extended to support polygonal agents. The vertices of polygonal agents query their environment like the other agents do. The edges of polygonal agents are registered as described hereafter. This way, each agent – point-like or polygonal – is able to take into account polygonal agents transparently.

4.2.1. Registering Edges.

The registration of an edge consists in spanning a particle [22] from one vertex to the other to gather the faces of the environment that have been crossed. The edge is registered as contained in all those faces.

The vertices and edges are registered so that their force fields can be considered by other agents. For that purpose, every agent retrieves registered entities from their current face and its one-ring (in blue in figure 5), i.e. its immediate vicinity.

Let us point out that in a typical simulation, agents stay in the same cell during many time steps. That means that the topological traversals used to retrieve neighborhood information frequently return the same data. Following this observation, proximity queries can be optimized by using spatial caching.

The registration process is adapted this way. Vertices are registered as contained in their current faces and as neighbor in their one-rings. Similarly, the edges are registered as contained in the sets of faces they cross and as neighbor in the one-rings of those sets of faces (figure 5).

This way, retrieving all interacting entities in the vicinity of an agent is performed in constant time and requires only one memory access to the data registered in its current face.

4.3. Updating Registrations

The registration information must be updated as soon as the faces in which vertices and edges lay change. For a point-like agent or vertex, this happens when its crosses edges of the environment. In this case, the vertex is unregistered from the face it quits and its one-ring and registered in the face it enters and its one-ring.

The edges of polygonal agents are processed the same way. For each edge, the sets of faces its crosses before and after the displacement are compared. If the crossed faces are the same, no update is need. In the other cases, the edge is unregistered and registered in its new position.

4.3.1. Handling Multiresolution.

At each time step and for every agent the following actions are performed. The set of vertices and edges in the neighborhood of the agent is retrieved. The distances from the agent to each element of this set are computed. The elements that are under the radius of influence are selected and their interactions are computed. Usually, to further reduce the computations, the elements are sorted by decreasing distances and only the nearest ones are taken into account. Optimizing this filtering process is one of the key issues to achieve real time simulations.

Large environment cells imply a higher computational cost of the filtering, in particular in dense situations. Smaller cells increases the probability for agents to cross edges and thus generates costly registration updates.

Subdividing the cells according to the density of agents is a good trade-off that efficiently minimizes these two costs (figure 6). Thus, whenever the number of agents in a cell exceeds a given threshold, it is subdivided. This way, the number of neighbors for each agent is bound which strongly reduces the cost of the filtering process. When the
density decreases, the subdivided faces are coarsened to maintain a reduced memory footprint and limit the updates.

To ensure that all agents in the interaction distance are taken into account, a minimum size is imposed on each face. Thus, faces are kept larger than the radius $R$ of influence of agents.

When a cell is subdivided, the registration information on the agents it and its adjacent faces contain has to be updated. Every agent is unregistered as contained before the subdivision and registered again after. Then, for each sub face the neighbors list is built by gathering the agents contained in its new one-ring.

The subdivision process is quite heavier for polygonal agents than for point-like agents. It produces smaller faces which, in turn, leads to larger sets of crossed cells for each edge. However, as will be shown in section 5, the adaptive subdivision of the environment significantly improves of global performance. The cost related to the updates is negligible in comparison with the observed gains.

4.3.2. Optimized Force Fields Evaluation.

A straightforward evaluation of the forces generated by a polygon at a given point would require $O(n)$ operations where $n$ is the number of vertices of the polygon. The multiresolution approach significantly improves that point.

In dense situations, the faces in which a polygonal agent $P$ is registered are subdivided, leading to a natural and adaptive decomposition of this agent. Hence, the agents moving in its vicinity only see the edges of $P$ that lie in the same face and its 1-ring.

Precisely, the proximity queries return the set of edges of polygonal agents that are registered in the actual face of an agent. Only the force generated by these edges (3 on average) are considered. The other edges are by construction at distances greater than their influence radius and would thus have null contributions.

As a consequence, the cost of the evaluation of the force fields generated by a possibly concave polygon depends, in dense scenarios, neither on the size of the polygon nor on the number of points of its boundary. This is an essential advantage to achieve real time performances.

4.3.3. Anticipation of Fast Agents.

Taking the velocity of agents into account is important to enhance the realism of collision avoidance. Fast agents should be visible a longer way ahead to reflect their influences on the trajectories of slower ones. As proximity query methods are only based on positions, they can not take velocities into account without a loss of efficiency.

The following extension addresses this issue. A vertex $h_1$ is added inside every polygonal agent, positioned either at its centroid or at the centroid of a subset of its vertices chosen to represent its head. A second vertex $h_2$ is positioned at $h_2 = h_1 + \Delta t \hat{v}$ where $\hat{v}$ is the mean velocity of the head. Segment $(h_1, h_2)$ approximates the move of the head during a time $\Delta t$. This segment is registered in the same way as the edges and generates a similar ellipsoidal force field. It makes the other agents move out of its trajectory. Figure 7 shows an example of a polygonal agent and its additional segment.

5. EXPERIMENTATION

The work presented here enriches the approach of [1] with the support of polygonal agents. Without this enrichment our framework would show similar performances than [1] that has already been compared to the $kd$-tree and spatial hashing [21] approaches. Thus we present a formal study of the time complexity of the different approaches, fol-
allowed by benchmarks to evaluate the cost of polygonal agents compared to simulations with point-like agents only.

5.1. Time Complexity

We evaluate an average time complexity for the querying, the building and the updates of the data structure that supports proximity queries. For that purpose, we denote: \( n_a \) the number of point-like agents ; \( n_p \) the total number of edges of polygonal agents ; \( n_f \) the number of edges of fixed obstacles ; \( n_c \) the average number of cells where a polygonal agent is registered ; and \( n_n \) the average number of neighbors in the visibility radius (closely related to the density). The number of features (vertices and edges) to register is \( n = n_a + n_p + n_f \).

We compare our multiresolution approach (MR) with methods using trees (kd-trees, BSP), Voronoi diagrams and regular grids (or spatial hashing). Building or updating the data structures at each time step has a time complexity in: \( O(n_a \log(n)) \) for trees and \( O(n_p) \) for grids and MR. Querying the data structure for the neighbors of an agent has a time complexity in: \( O(n_a \log(n)) \) for trees ; \( O(n_p) \) for Voronoi diagrams ; and \( O(1) \) for grids and MR.

The overall time complexity of our method is in \( O(n) \) versus \( O(n \log(n)) \) for classical ones.

5.2. Benchmarks

We set up several scenarios to test our system in different configurations (figure 8).

In Crossing-N-M, \( N \) point-like agents navigate from top to bottom or in the opposite direction, while \( M \) polygonal agents navigate from left to right or in the opposite direction.

In Circle-N-M, \( N \) point-like agents are distributed along a circle and walk towards the diametrically opposed position while \( M \) polygonal agents move on two circles in the center of the scene.

We also evaluated our method in non-planar scenarios, as shown in figure 4, but such results are not presented here.

In the following, we try to evaluate the computation times of the behavior model, the proximity queries and the multiresolution registration. Other tasks, like rendering or path planning are not taken into account.

We measured the time spent for the proximity queries on the Crossing-N-M scenario with an increasing number of polygonal agents. Figure 9 shows that the computation time increases linearly with the number of polygonal agents. On average, a proximity query is thus achieved in constant time even if agents agglutinate around the...
Deformable polygonal agents in crowd simulation

Figure 10. Computation times during 10000 time steps with varying N (points) and M (polygons). Left: Circle-N-M scenario; Right: Crossing-N-M scenario with N=1000 and M=50.

polygonal agents. That confirms that the multiresolution approach effectively bounds the number of neighbors.

To precisely analyze the performance of our approach, we detail the costs of the different tasks of the simulation: registration updates for point-like and polygonal agents, subdivision and coarsening operations required by the multiresolution management and behavior computation. Results are shown in figure 10.

It shows that the cost of multiresolution is negligible compared to the registration updates. The left chart shows that the computation cost for point-like agent is not influenced by the number of polygonal agents. Conversely, the cost of the polygonal agents' registration does not depend on the density of point-like agents.

The right chart shows computation times for the Crossing-N-M scene using three distinct representations for the environment. The first and second benchmarks use static regular grids respectively at low and high resolutions. The third measurement shows the results with our multiresolution mechanism that adapts the size of the faces to the crowd density.

The measurements clearly show that the cost of the point-like agents' management – here the proximity querying – is reduced for finer grid and for our multiresolution approach. That confirms that the cost of the filtering process increases with the size of cells. On the other hand, smaller faces increase the cost of polygonal agents registration, as it increases the number faces crossed by their edges and thus the complexity of the registration operation.

Our multiresolution approach efficiently manages both cases and provides a good trade-off between the handling of polygonal agents and the adaptation of the environment mesh to the density of the crowd. In practice, as polygonal agents repulse point-like ones, their surrounding cells are quite empty which triggers the coarsening process of the multiresolution map and allows to keep the complexity of the registration operation stable even with a globally large number of agents.

6. CONCLUSION

In this paper, we have proposed a crowd simulation framework that handles both point-like and polygonal agents in real time. It supports agents whose shape is modeled by arbitrary deformable polygons. This includes vehicles, articulated objects or 3D cages commonly used to handle animated characters or deformable objects.

Point-like and polygonal agents interact thanks to efficient proximity queries. We demonstrate the effectiveness of the registration process in a multiresolution framework, even for complex shapes. The subdivision of the environment is controlled according to the local density of the crowd. Wide cells in sparse areas lead to few registration updates. Small cells in dense areas limit the complexity of proximity queries.

We have described a potential field behavior model adapted to polygonal, possibly concave, agents. We also introduced a technique to take into account the velocity of agents in proximity queries that fits naturally in our framework.

Future work will focus on the improvement of path finding by exploiting the density information contained in the multiresolution representation. The polygonal representation of agents – or cages – could also be refined or coarsened dynamically according to the needs of the behavior model or to the desired precision of the simulation. A definition of the RVO behavior model adapted to polygonal agents may also offer interesting perspectives.

ACKNOWLEDGEMENTS

This work has been founded by Région Alsace and French CNRS.

REFERENCES


**AUTHORS’ BIOGRAPHIES**

**Thomas Pitiot** is a second year PhD student from ICube Lab at the Université de Strasbourg, Strasbourg, France. He received both his Master in Computer Science from the Université d’Evry Val d’Essonne and his Engineer diploma from ENSIIE in 2012. His main research interests include animation, collision detection and crowd simulation.

**David Cazier** is an assistant professor in Computer Science at Université de Strasbourg, France, and a researcher at the ICube Lab since 1999. He received his PhD in Computer Science from the Université d’Evry Val d’Essonne and his Engineer diploma from ENSIIE in 2012. His main research interests include geometric modeling and real-time simulations. He develops new multi-resolution approaches and tools for collision detection and collision avoidance in computer animations with applications in crowd simulations and augmented reality for surgical planning and for intra-operative assistance.

**Thomas Jund** is a researcher in Computer Science at InSimo, Strasbourg, France, and a researcher at the ICube Lab since 1999. He received his PhD in Computer Science from the Université de Strasbourg in 1997. His research interests include surgery simulation, animation and computational geometry based on topological models.
Arash Habibi is a senior computer graphist and an assistant professor in Computer Science at the Université de Strasbourg and a researcher at Icube. His research interests include dynamic models and procedural animation such as for crowds.

Pierre Kraemer is an assistant professor in Computer Science at Université de Strasbourg, France, and a researcher at the ICube Lab since 2009. He received his PhD in Computer Science from the Université de Strasbourg in 2008 and spent a year as post-doctoral fellow at INRIA Nancy Grand-Est. His main research interests include geometric modeling based on topological models, multiresolution representations and geometry processing.